



# Rapport TD1 MI206 : Géométrie Algorithmique et Morphologie mathématique

Bo LI

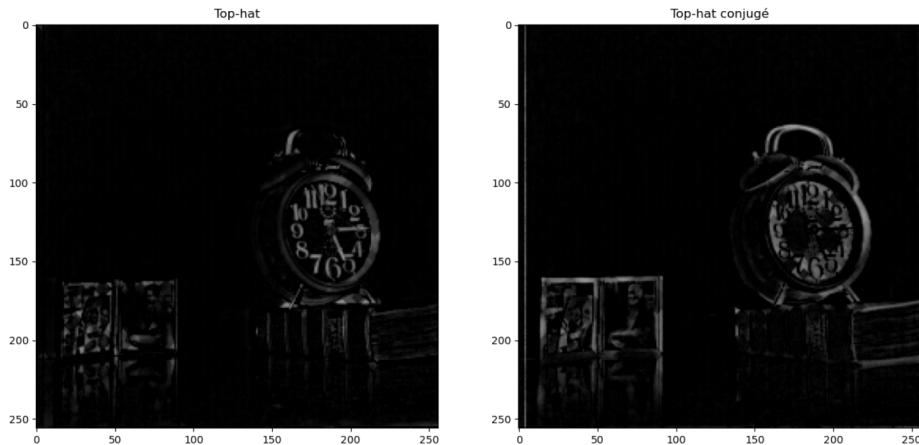
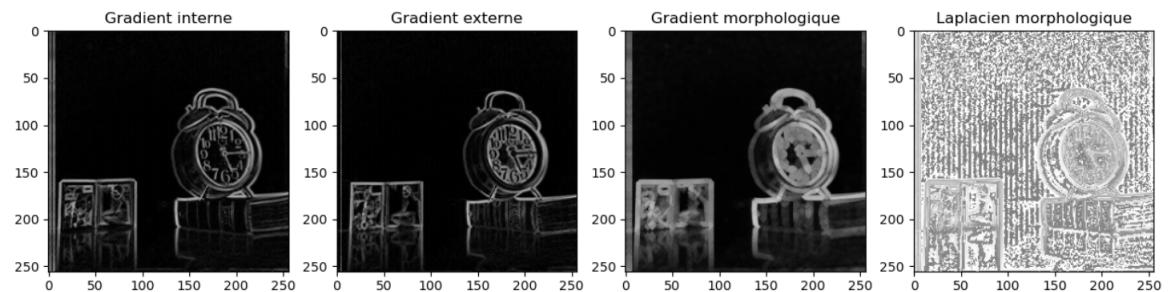
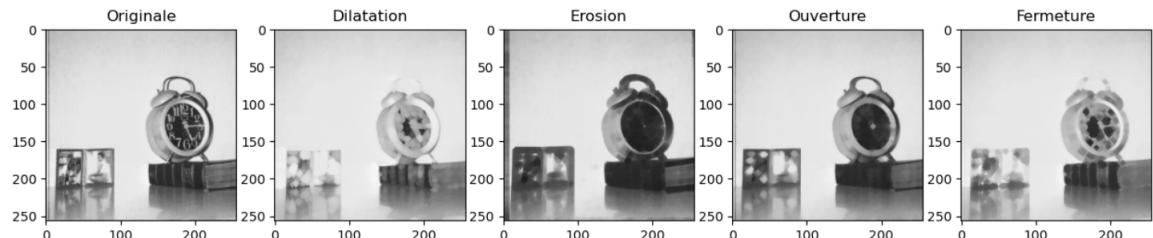
06/05/2020

# Table des matières

<b>1</b>	<b>Opérateurs de base et composés</b>	<b>2</b>
<b>2</b>	<b>Un détecteur de contours rudimentaire</b>	<b>3</b>
<b>3</b>	<b>Transformée en Tout-ou-rien</b>	<b>8</b>
<b>4</b>	<b>Opérateurs connexes</b>	<b>8</b>
4.1	algorithme de reconstruction . . . . .	8
4.2	Application de reconstruction . . . . .	9
4.3	érosion ultime . . . . .	10
<b>5</b>	<b>Lignes de Partage des Eaux</b>	<b>11</b>
5.1	filtre spatial . . . . .	12
5.2	filtre dynamique . . . . .	13
5.3	filtre combiné . . . . .	13
<b>6</b>	<b>Opérateur de Contraste</b>	<b>14</b>
<b>7</b>	<b>Nivellement</b>	<b>17</b>
7.1	FAS . . . . .	18
7.2	FAS par reconstruction . . . . .	19

# 1 Opérateurs de base et composés

Je prends l'image de réveil comme data à traiter. les opérations morphologiques sont présentées ci-dessous.



Pour vérifier expérimentalement la relation de dualité, il faut vérifier l'équation suivante par définition :

$$\boxed{\Phi(\bar{x}) = \overline{\Phi^*(x)}}$$

Je vérifie cette équation pour erosion/dilation et ouverture/fermeture :

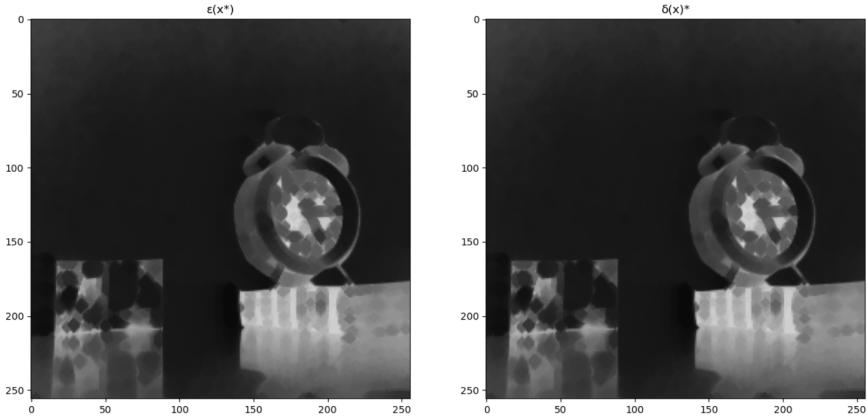


FIGURE 1 – erosion/dilation

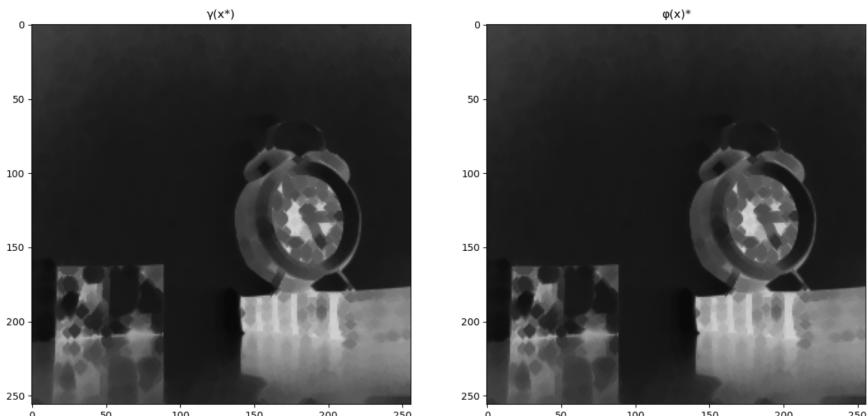


FIGURE 2 – ouverture/fermeture

On peut remarquer que chaque paire est identique.

## 2 Un détecteur de contours rudimentaire

Selon le modèle classique de Marr et Hildreth, Je construis un détecteur de contours rudimentaire. Je l'applique sur plusieurs images.

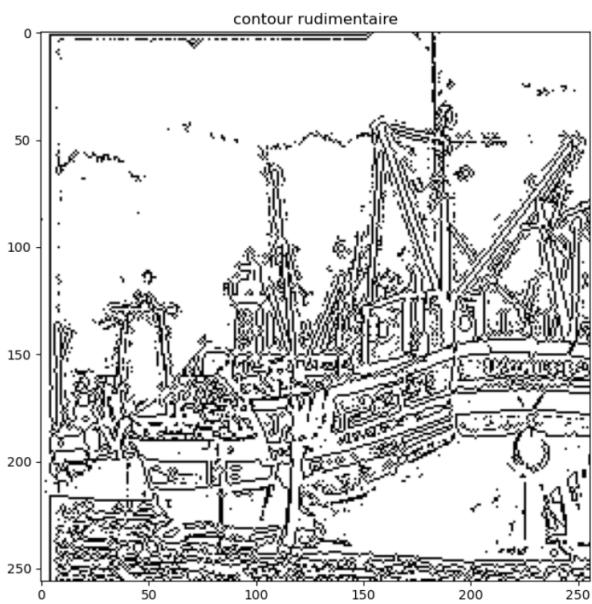


FIGURE 3 – boats s=15

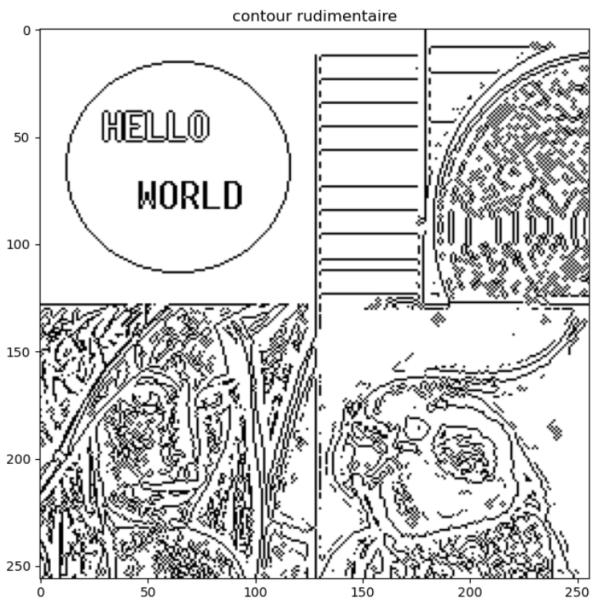


FIGURE 4 – montage s=15

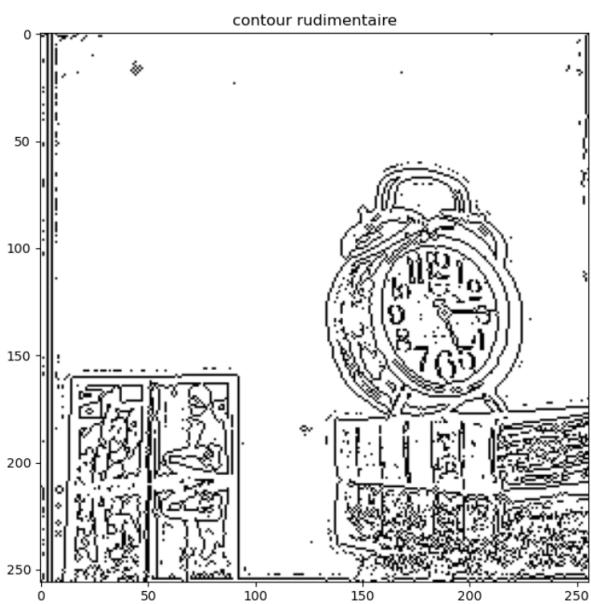


FIGURE 5 – clock s=15

Le détecteur marche bien sur ces images, les contours sont bien marqués. Les paramètres sont l’élément structurant et le seuillage  $s$ . Plus le seuillage est grand, le résultat est meilleur. Si on diminue le seuillage, le résultat est moins bon. Voir les exemples :

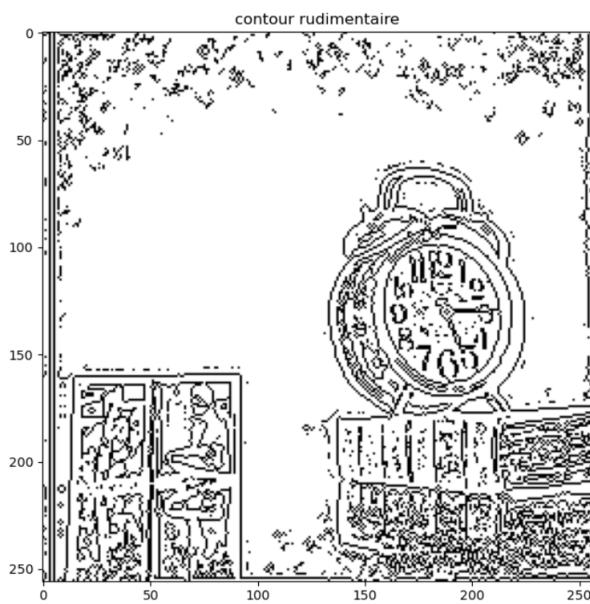


FIGURE 6 – clock s=10

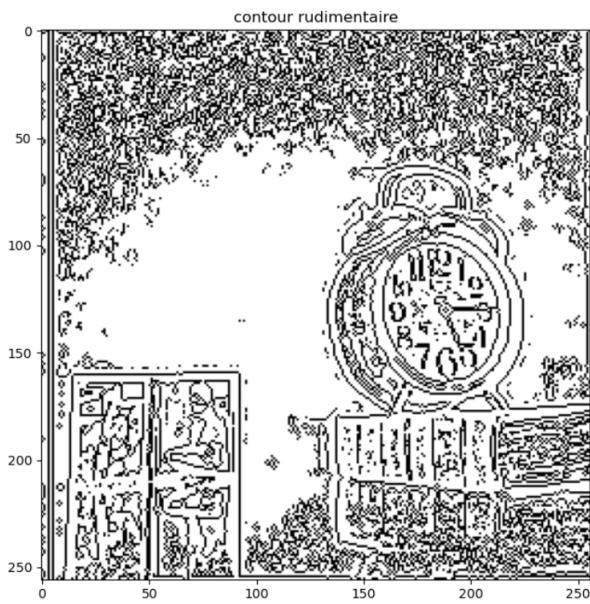


FIGURE 7 – clock s=5

Mais s'il est trop grand, certains contours sont effacés. Ici, J'applique  $s = 50$ .

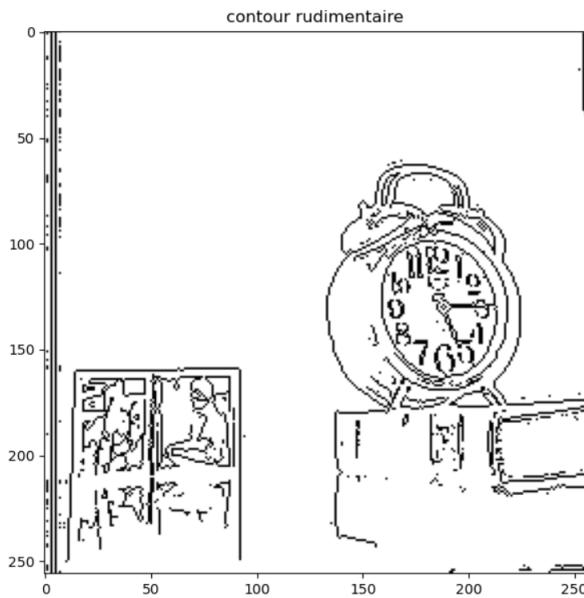


FIGURE 8 – clock s=50

Donc, il faut bien choisir le seuillage pour obtenir le meilleur résultat. Si l'image est plus compliquée, on doit augmenter le seuillage.

Le code :

```

seuillage = 15
gammaplus = np.zeros((256,256))
gammaminus = np.zeros((256,256))
Gs = np.zeros((256,256))
imDil = dilation(img,se4) # Dilatation morphologique
imEro = erosion(img,se4) # Erosion morphologique
imgrai = img - imEro
imgrae = imDil - img
for i in range(256):
    for j in range (256):
        if imgrae[i][j] > imgrai[i][j]:
            gammaplus[i][j] = 1
        else:
            gammaminus[i][j] = 1
        if imgm[i][j] >= seuillage:
            Gs[i][j] = 1
gammazero = dilation(gammaplus,diamond(1))

```

```

for i in range(256):
    for j in range (256):
        if Gs[i][j] == 1:
            if(gammazero[i][j]==0 | (gammaminus[i][j]
                ]==0):
                Gs[i][j] = 0

```

### 3 Transformée en Tout-ou-rien

Le transformée en Tout-ou-rien sert au débruitage. Il est réalisé par l'équation suivante :

$$I \circledast (H, M) = \varepsilon_H(I) \cap \varepsilon_M(I^c) = (I \ominus H) \cap (I^c \ominus M).$$

Ici, il n'y a pas d'intersection entre H et M. Je l'applique sur une image bruitée :

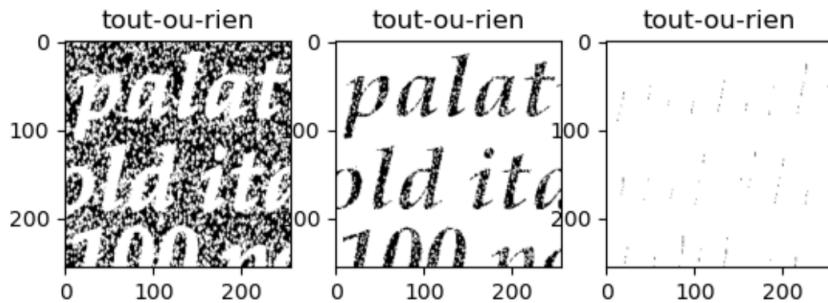


FIGURE 9 – Transformée en Tout-ou-rien

Le code :

```

imginverted = (imgBin == False)
imgeroH = erosion(imgBin,H)
imgeroM = erosion(imginverted,M)
img_tout_ou_rien = imgeroH & imgeroM

```

### 4 Opérateurs connexes

#### 4.1 algorithme de reconstruction

La reconstruction est réalisée par :

$$E_{B_1}^R(X) = \sup_{n \geq 0} \left\{ (\delta_{B_1}^R)^n(X) \right\}$$

Je la programme en python :

```
def RECONSTRUIT (img,img_ini):
    size = np.size(img[0])
    imtmp = np.tile(img,1)
    for k in range(20):    ### r p ter jusqu' stable
        imtmp = dilation(imtmp,disk(2))
        for i in range(size):
            for j in range(size):
                imtmp[i][j] = min(img_ini[i][j],
                                   imtmp[i][j])
    return imtmp
```

## 4.2 Application de reconstruction

J'applique la reconstruction sur l'image *particules.png*. Pour éliminer les particules de diamètre inférieur à 20 pixels. Il faut appliquer l'ouverture par reconstruction et choisir un élément structurant de diamètre 20 pixels.

Le résultat :

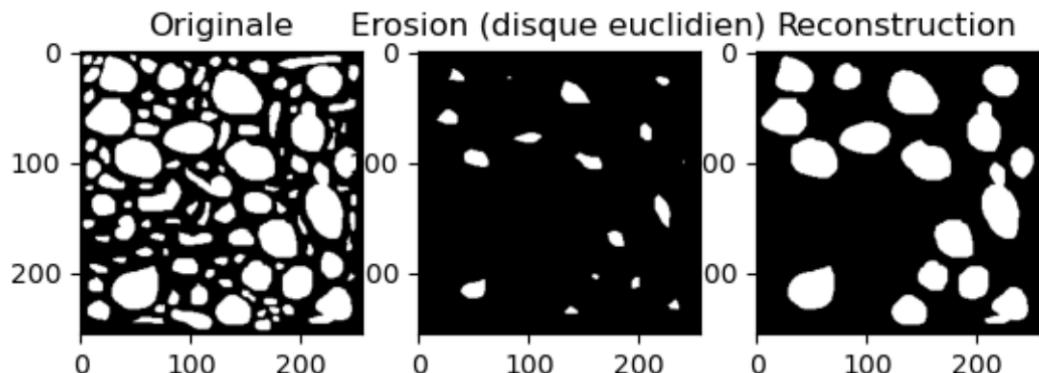


FIGURE 10 – Élimination des particules inférieur à 20 pixels

J'applique la reconstruction sur l'image bruitée *tree\_noise.jpg*.

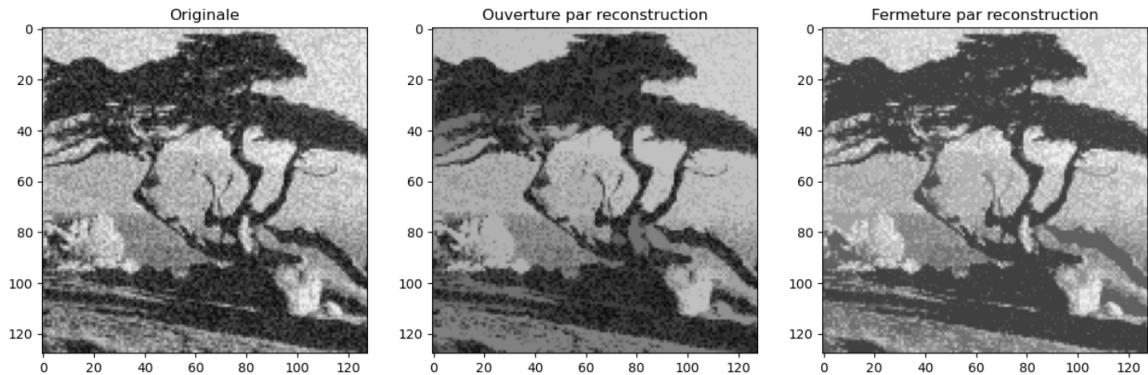


FIGURE 11 – reconstruction sur image bruitée

Les constructions peuvent aider au débruitage. On peut aussi remarquer que l’ouverture par reconstruction garde mieux les parties sombres et la fermeture par reconstruction garde mieux les parties brillantes.

### 4.3 érosion ultime

L’érosion ultime se définit comme le *residu* entre érosion et les ouvertures par reconstruction de chaque érosion dans la précédente. Mathématiquement, il est défini par :

$$U_i(X) = \epsilon_{iB}(X)/\gamma^{Rec}(\epsilon_{(i+1)B}(X) : \epsilon_{iB}(X))$$

Je réalise un algorithme en python :

```
def EROSION_ULTIME(img):
    imX = np.tile(img,1)
    size = np.size(img[0])
    imEroU = np.zeros((size,size),dtype = int)
    for k in range(10): ##repeter jusqu' stable
        imtmp = np.tile(imX,1)
        imX = erosion(imX, diamond(2))
        imY = RECONSTRUIT(imX, imtmp)
        Utmp = imtmp - imY
        for i in range(size):
            for j in range(size):
                if Utmp[i][j] == 1:
                    imEroU [i][j] = 1
```

```
    return imEroU
```

Je l'applique sur *coffee.png* :

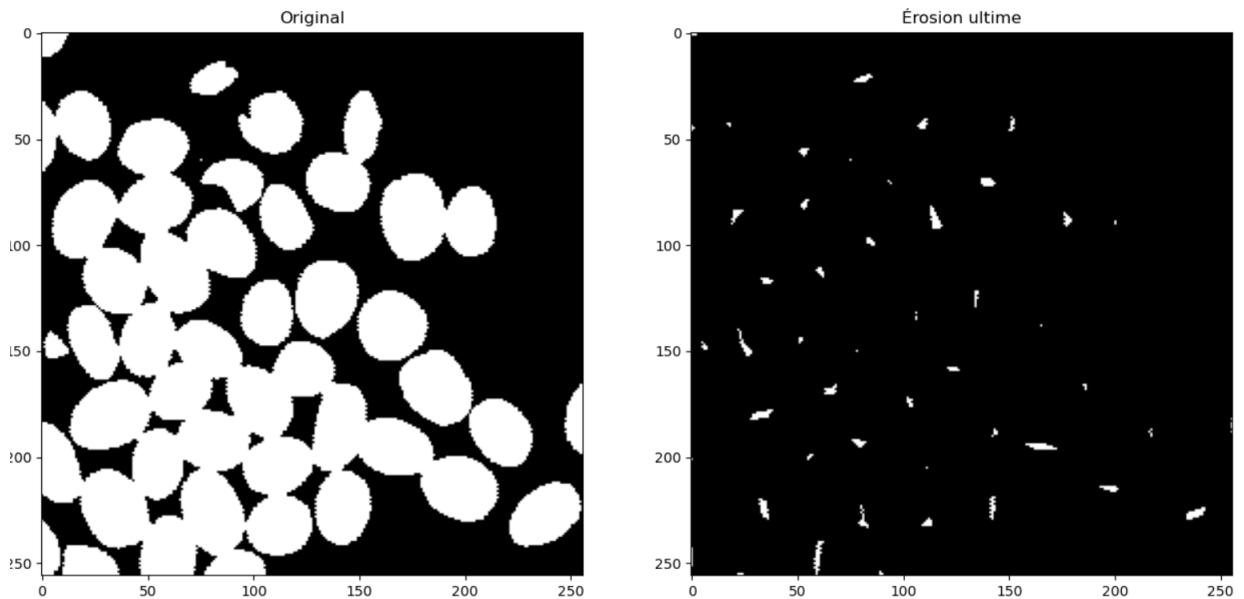


FIGURE 12 – érosion ultime

L'algorithme marche bien et trouve l'ensemble des points qui reste après une érosion ultime.

## 5 Lignes de Partage des Eaux

J'applique l'opérateur de lignes de partage des eaux sur l'image *uranium.png*.

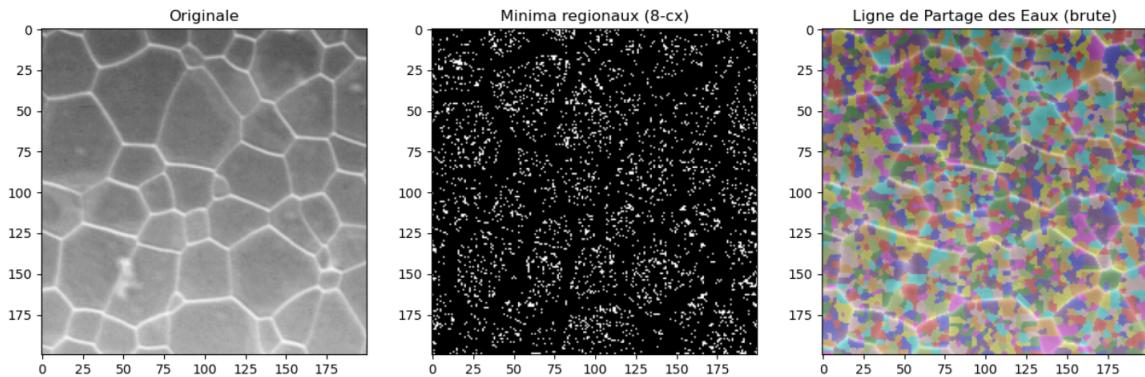


FIGURE 13 – LPE brute

La LPE a pour but d'obtenir une partition de l'image en régions regroupant des pixels jugés équivalents en fonction d'un certain critère. Généralement, Je l'applique sur les images unnaturalles. Dans le cas d'images naturelles, la LPE produit une sur-segmentation très importante.

L'application de LPE sur *uranium.png* conduit à une sur-segmentation. Je donne un autre exemple :

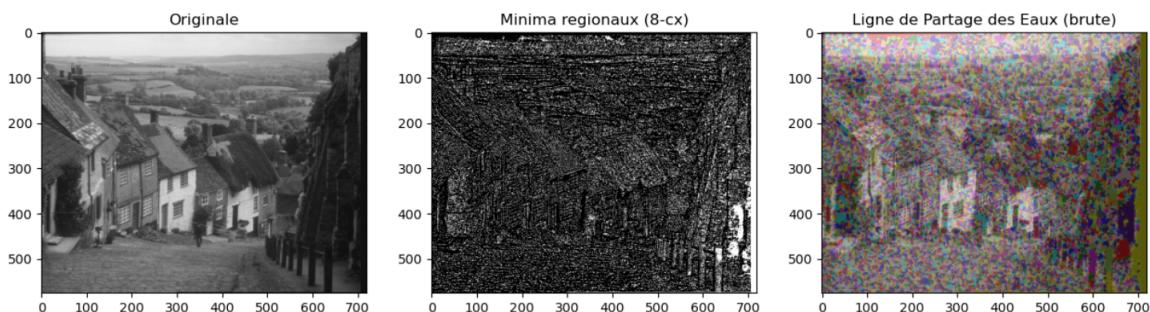


FIGURE 14 – sur-segmentation

Pour résoudre ce problème, on doit appliquer les filtres sur l'image avant LPE.

## 5.1 filtre spatial

J'applique une fermeture par reconstruction sur l'image de gradient avant la LPE. Cette technique peut éliminer des digues produites par des bassins de faible surface. Le résultat est présenté ci-dessous :

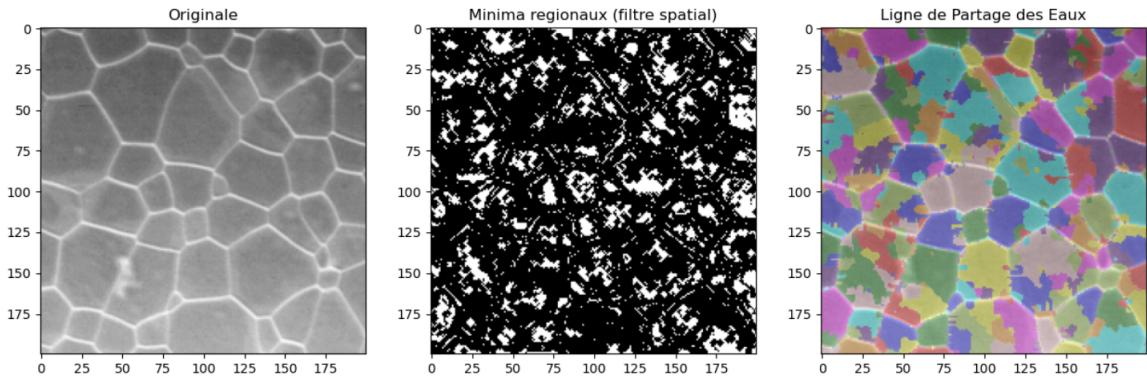


FIGURE 15 – filtre spatial

La sur-segmentation est réduite, mais pas totalement.

## 5.2 filtre dynamique

J'applique un filtrage de dynamique sur l'image originale avant le calcul du gradient. Il est réalisé par une construction de fonction de niveau en gris  $I$  dans la fonction  $I + h$ . Cette technique peut éliminer des digues produites par des bassins de faible dynamique. Le résultat est présenté ci-dessous :

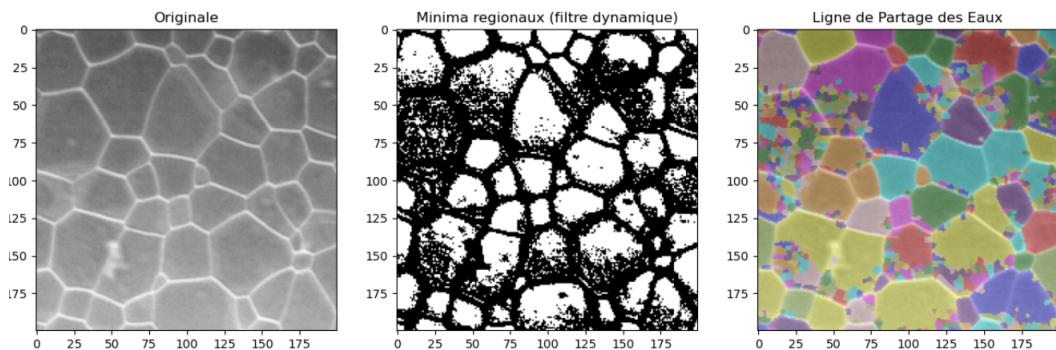


FIGURE 16 – filtre spatial

La sur-segmentation est réduite, mais pas totalement.

## 5.3 filtre combiné

Si on combine ces deux filtres, on peut obtenir le résultat ci-dessous :

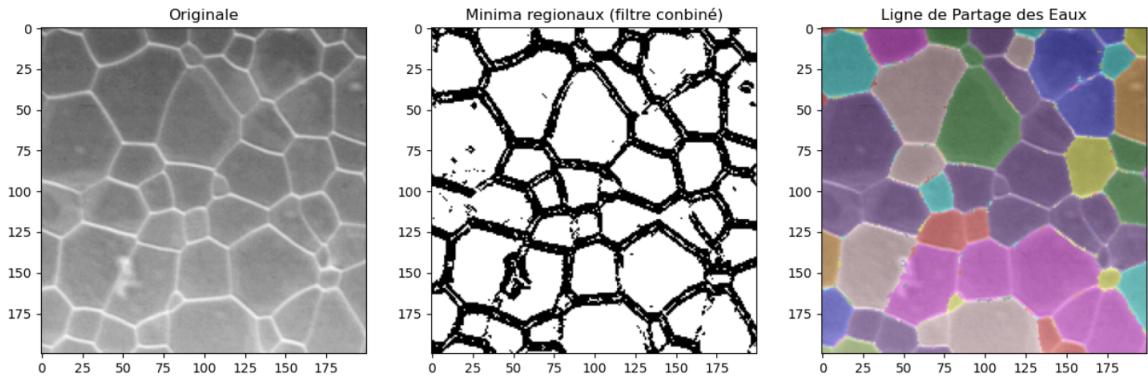


FIGURE 17 – filtre combiné

La segmentation est presque parfaite sur image, les blocs sont bien identifiés et séparés.

## 6 Opérateur de Contraste

L'opérateur de contraste est réalisée par l'équation suivante :

$$\chi(x) = \arg \min_{\omega \in \{\xi, \psi\}} \{|\omega(x) - x|\}(x)$$

Je implemente cette opérateur en python, le code est présenté à la fin.

Voici les résultats en utilisant trois types d'opérateurs :

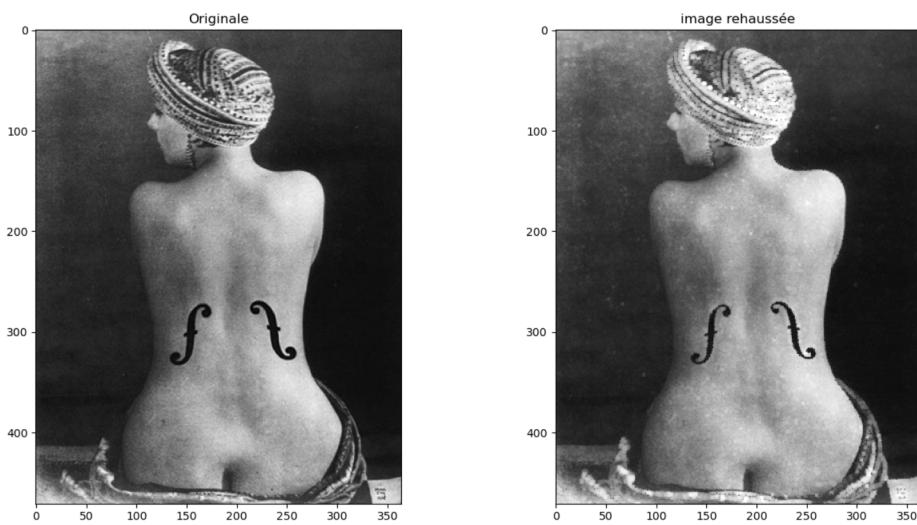


FIGURE 18 – par (érosion, dilatation)

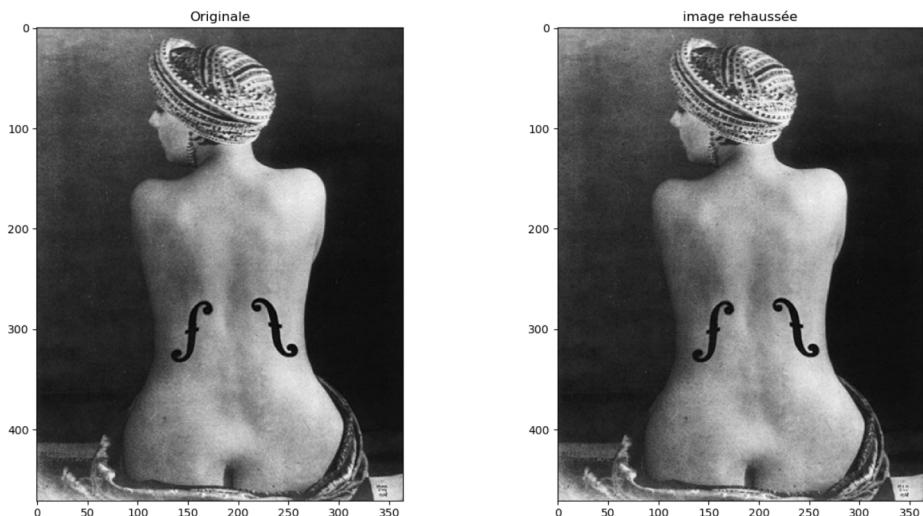


FIGURE 19 – par (ouverture, fermeture)

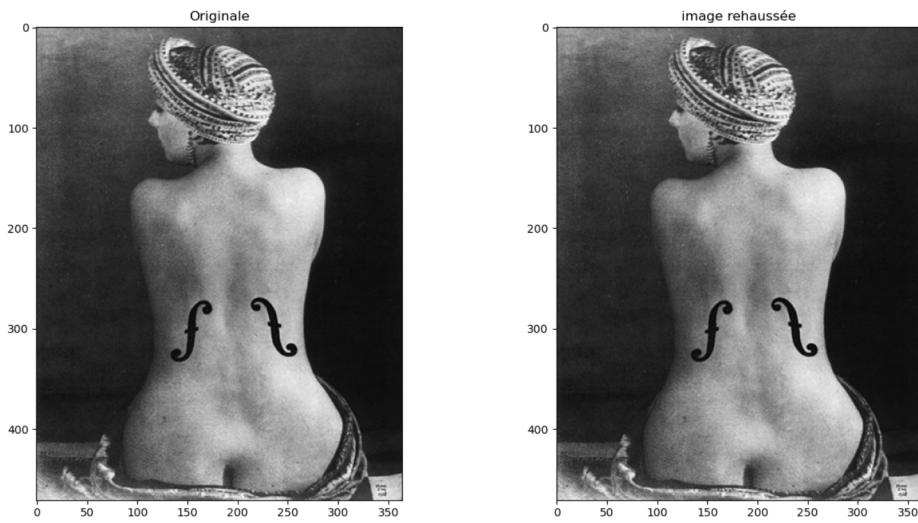


FIGURE 20 – par (ouverture, fermeture) par reconstruction

```

def Contraste(image ,type):
    str_size = int(input("ins rer la taille de l'\
        lment      structurant\n"))
    correct = False
    while(correct==False):
        str_type = input("choisir un type de de l'\
            lment      structurant parmi disk, diamond et\
            square\n")
        if str_type == 'disk':
            str = disk(str_size)
            correct = True
        elif str_type == 'square':
            str = square(str_size)
            correct = True
        elif str_type == 'diamond':
            str = diamond(str_size)
            correct = True

    size1 = image.shape[0]
    size2 = image.shape[1]
    imcontrast = np.zeros((size1, size2))
    if type == 0:
        im1 = erosion(image,str)
        im2 = dilation(image,str)

```

```

    elif type == 1:
        im1 = dilation(erosion(image,str),str)
        im2 = erosion(dilation(image,str),str)
    elif type == 2:
        im1 = reconstruction(erosion(image,str),image)
        im2 = reconstruction(dilation(image,str),image,
                             method = 'erosion')
    else :
        return "please choose type from 0 1 and 2"

    for i in range(size1):
        for j in range(size2):
            if abs(im1[i][j] - image[i][j]) < abs(im2[i]
                ][j] - image[i][j]):
                imcontrast[i][j] = im1[i][j]
            else:
                imcontrast[i][j] = im2[i][j]
    return imcontrast

```

## 7 Nivellement

Le nivelllement peut être vu comme une reconstruction mixte. Il est défini par :

$$P(m, c) = \rho(m_{\leq}, c_{\leq}) \cup \rho'(m_{\geq}, c_{\geq})$$

Les deux FAQs sont définis par :

$$\Theta_{rayon} = \phi_{rayon} \gamma_{rayon} \dots \phi_1 \gamma_1$$

$$\Xi_{rayon} = \gamma_{rayon} \phi_{rayon} \dots \gamma_1 \phi_1$$

Je réalise ces opérateurs en Python, les codes sont présentés à la fin.

J'applique la FAS sur *goldhill.png* :

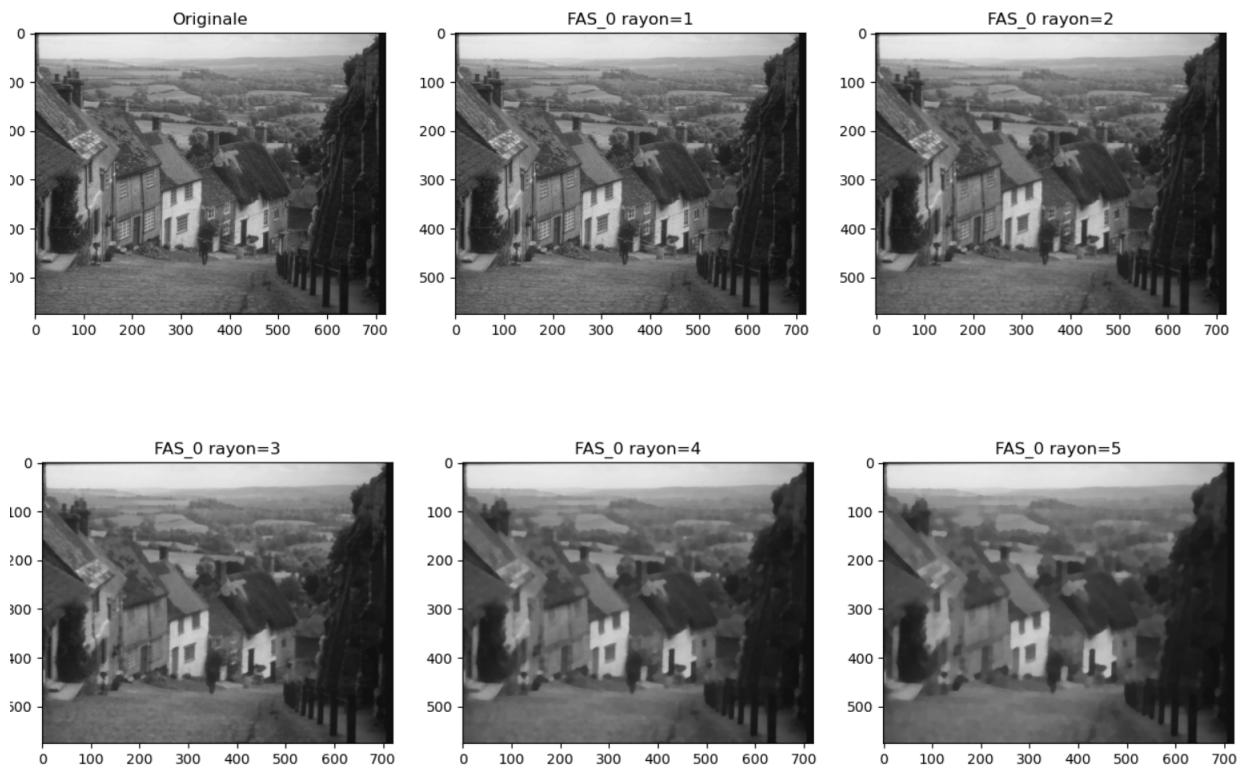


FIGURE 21 – FAS d’abord ouverture avec rayon = 1, 2, 3, 4, 5

FAS fournit une représentation des images à différents niveaux de détail. Plus le rayon est grande, plus le niveau de détail est petit.

FAS peut servir comme une bonne réduction du bruit grâce à une élimination progressive des pics et des creux de faible surface.

## 7.1 FAS

Je l’applique sur une image bruitée :

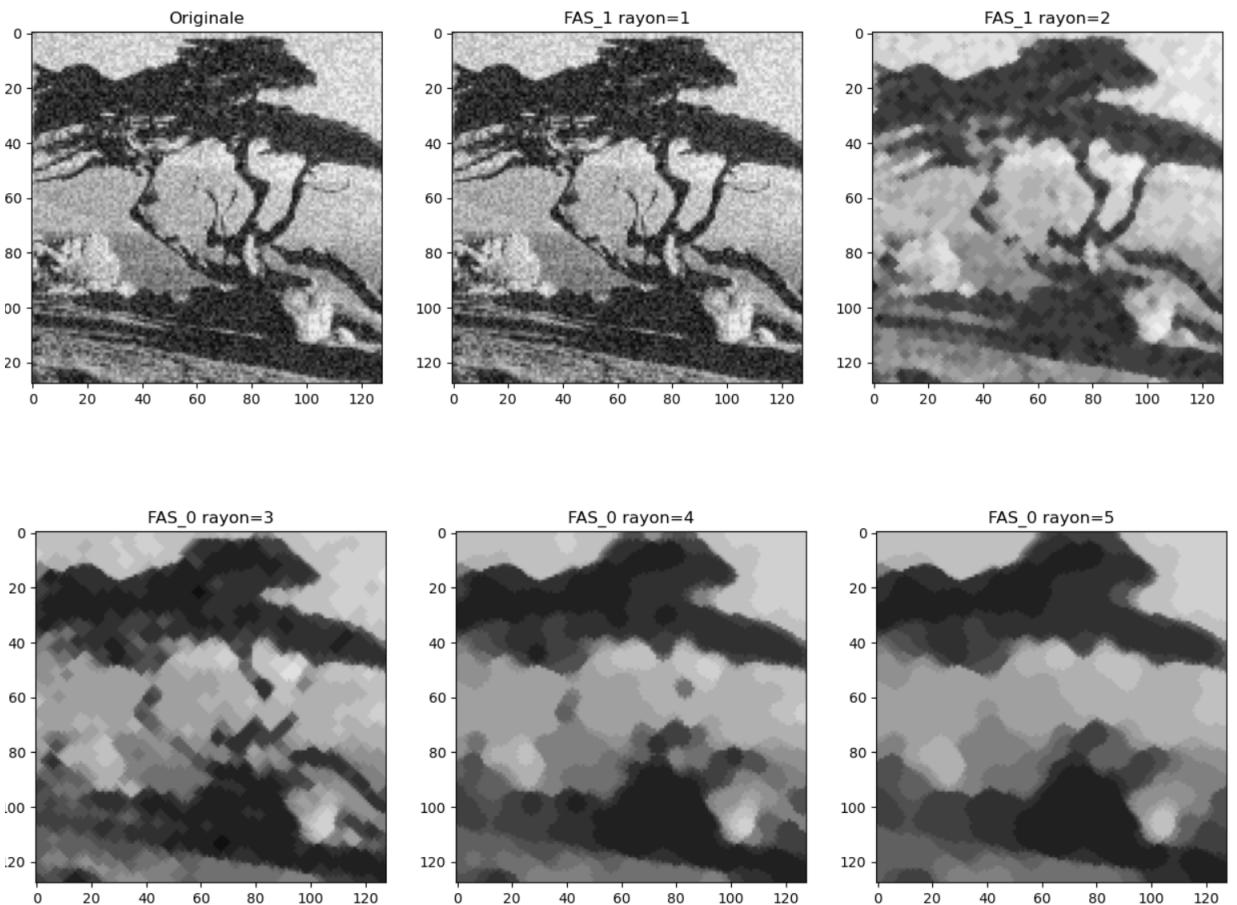


FIGURE 22 – FAS sur une image bruitée avec rayon = 1, 2, 3, 4, 5

## 7.2 FAS par reconstruction

FAS par reconstruction est le nivlement de FAS. On applique FAS sur les même images :





FIGURE 23 – FAS par reconstruction rayon = 1, 2, 3, 4, 5

Et sur une image bruitée :

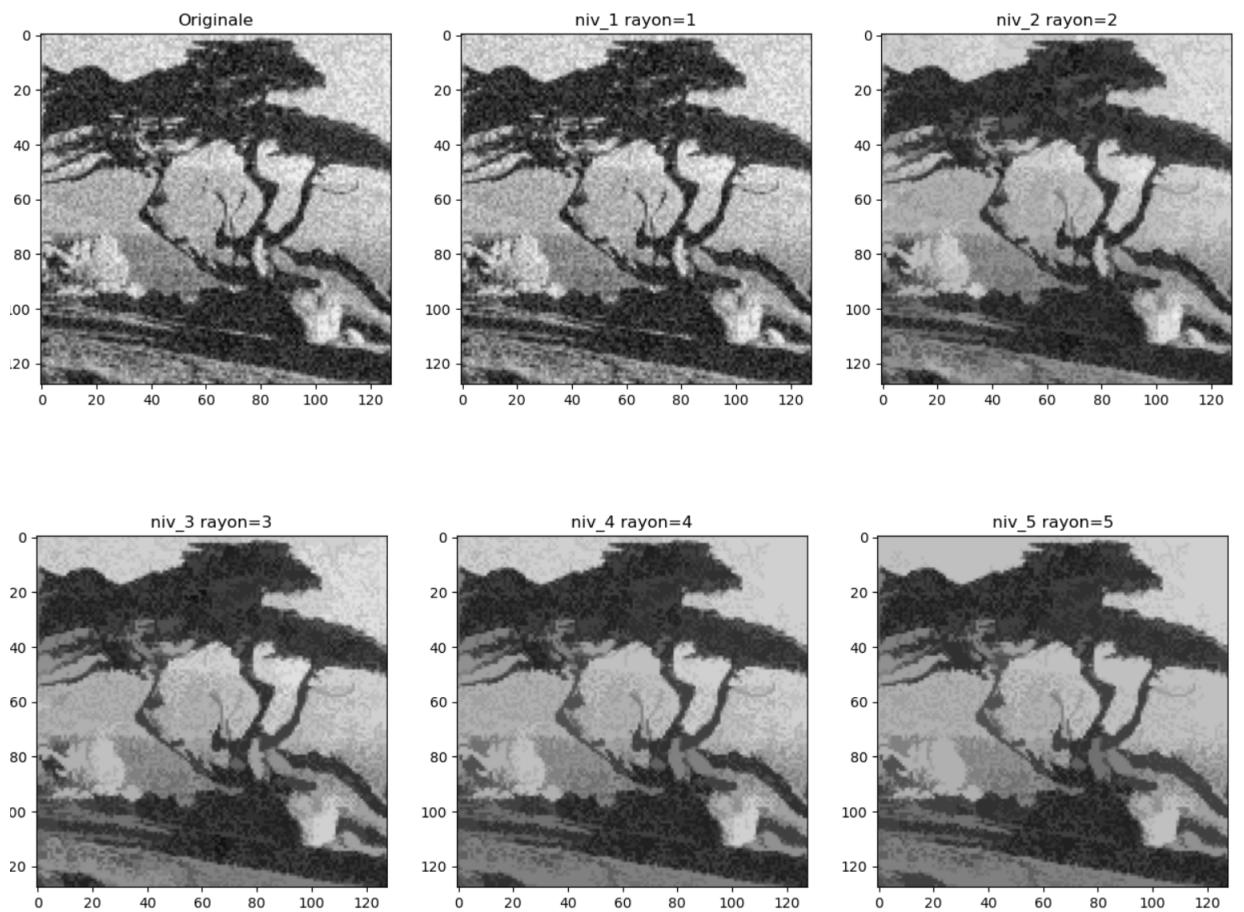


FIGURE 24 – FAS sur une image bruitée avec rayon = 1, 2, 3, 4, 5

On peut remarquer que FAS par reconstruction peut aussi servir au débruitage, mais le résultat est plus doux que celui de FAS.

les codes :

```
def FAS(image, premier, rayon):
    imtmp = np.tile(image, 1)
    if premier == 0:
        for i in range(rayon):
            str = disk(i)
            imtmp = dilation(erosion(imtmp, str), str)
            imtmp = erosion(dilation(imtmp, str), str)

    elif premier == 1:
        for i in range(rayon):
            str = disk(i)
            imtmp = erosion(dilation(imtmp, str), str)
            imtmp = dilation(erosion(imtmp, str), str)
    else:
        return 'premier should be 0 or 1'
    return imtmp

def Nivellement(image, image_ref, cx):
    if cx != 4 | cx != 8:
        return "cx should be 4 or 8"
    size1 = image.shape[0]
    size2 = image.shape[1]
    sup = np.zeros((size1, size2)) ##image sup rieure
    inf = np.zeros((size1, size2)) ##image inf rieure
    for i in range(size1):
        for j in range(size2):
            if image[i][j] < image_ref[i][j]:
                sup[i][j] = 255
                inf[i][j] = image[i][j]
            else:
                sup[i][j] = image[i][j]
                inf[i][j] = 0
    recon_inf = reconstruction(inf, image_ref)
    recon_sup = 255 - reconstruction(255-sup, 255-
        image_ref)
    return recon_inf + recon_sup
```